Appendix E. Remote Temperature Sensor Case Study

This appendix describes an application of the proof-of-concept prototype, CODA, described in Chapter 10, to a remote temperature sensor problem. The specification for this system consists of a single, data flow diagram and a textual description. The specification comes from Nielsen and Shumate.[1] [Nielsen88, Appendix A] This specification makes an interesting case study because Nielsen and Shumate use Structured Analysis, augmented with event flows, in lieu of RTSA, to model the problem. This choice limits the semantic model of the remote temperature sensor in two ways. First, the data flow diagram, consistent with Structured Analysis, does not include control transformations and state-transition diagrams. Consequently, the specification also does not include triggers, enables, and disables. Second, the data flow diagram does not take advantage of many of the modeling capabilities inherent in COBRA. Instead, Nielsen and Shumate perform a functional decomposition that leads to several chains of transformations, where each transformation represents an aperiodic function. These facts mean that: 1) CODA must generate a design from a data flow diagram that uses only a subset of the semantic concepts included within the specification meta-model and 2) CODA must reason about chains of aperiodic functions to a degree not seen in the previous case studies. The

[1] The literature contains other treatments of this same problem. [Carter88, Cherry86, Howes90, Nielsen87, Sanden89a, Sanden94, Shumate92, Smith93, Young82]

remote temperature sensor application, then, provides an example where CODA's reasoning abilities provide the designer with only a limited degree of assistance.

E.1 Analyzing the Specification

Nielsen and Shumate provide a data flow diagram for a remote temperature sensor. [Nielsen88, page 278] A designer loads this diagram, exactly as drawn by Nielsen and Shumate, into CODA and then CODA analyzes the diagram. Figure 71 depicts the diagram, annotated, as in previous case studies, with the information inferred and elicited by CODA.

E.1.1 Evaluating the Original Data Flow Diagram

After loading the remote temperature sensor (RTS) specification, the designer examines the state and finds the condition of the specification to be unknown. Finding the classification of concepts to be incomplete, the designer asks CODA to classify concepts in the specification. After querying the designer about the nature of the terminators on the diagram, CODA proceeds through the first three stages of classification without consulting the designer. In the fourth stage of classification, CODA asks the designer for assistance to classify a number of transformations.

First, CODA identifies three transformations, Prepare CP ACK, Get Temperature Reading, and Maintain Temperature Table, likely to represent synchronous functions. CODA asks the designer to confirm or override these decisions. In all three cases, the designer confirms the classifications. Next, CODA encounters eight transformations, Determine Msg Type, Create ICP, Validate ICP, Monitor Periodic Query, Wait for DP

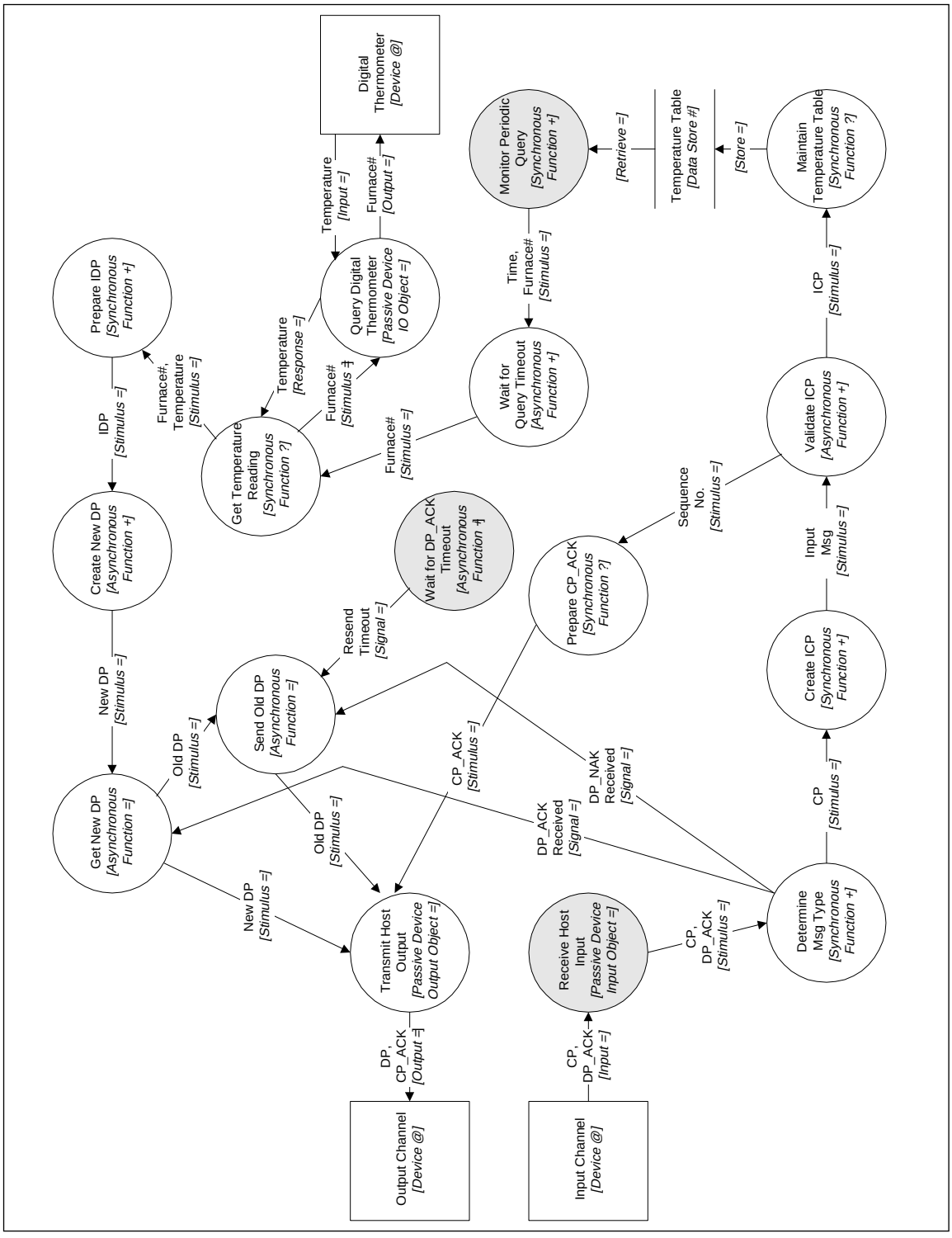

Figure 71. Original RTS Data Flow Diagram as Analyzed by CODA

ACK Timeout, Wait for Query Timeout, Create New DP, and Prepare IDP, that can be more accurately classified based on application-specific knowledge that might be available to the designer. In each of these eight cases, CODA consults the designer, who provides the requested information. Based on this information CODA classifies the transformations. After classifying all elements on the data flow diagram, CODA elicits any additional information that might help with design generation. In this example, the designer provides no additional information. Next, the designer attempts to verify the specification's utility. Unfortunately, though all specification elements are classified fully, some axioms remain unsatisfied.

E.1.2 Correcting and Reevaluating the Data Flow Diagram

By examining the notices logged by CODA, the designer finds that three transformations, shaded in Figure 71, violate axioms for concepts of their type. One transformation, Receive Host Input, classified as a Passive Input Device Object, violates two axioms. First, any Passive Input Device Object must receive a Stimulus or a Signal. Second, any Passive Input Device Object must emit a Response. Examination of Figure 71 reveals that the transformation, Receive Host Input, does indeed violate these axioms. Further, after reviewing the textual specification concerning this transformation, the designer realizes that the devices in the problem are all asynchronous devices, not passive devices, as classified by CODA. These problems arise because Nielsen and Shumate do not depict the event flows from the external devices to the appropriate transformations.

The designer corrects these deficiencies simply by adding three event flows, one from each Terminator to its corresponding Interface Object, to the data flow diagram.

Another transformation, Wait for DP ACK Timeout, classified as an Asynchronous Function, violates two axioms. First, any transformation requires at least one incoming arc. Second, any function requires an incoming activator, that is, a Signal, a Stimulus, a Timer, or a Control Event Flow. A review of Figure 71 verifies that the transformation in question, Wait for DP ACK Timeout, violates these two axioms. Upon reading the textual specification for the remote temperature sensor, the designer discovers that the transformation, Wait for DP ACK Timeout, serves only to generate a periodic event flow to another transformation, Send Old DP. The specification meta-model, allows this requirement to be modeled easily with a Timer event flow directly into the periodic transformation, Send Old DP, from the system. Here, then, the designer eliminates one transformation, Wait for DP ACK Timeout, from the data flow diagram, and changes the source for one event flow, Resend Timeout, to be "System".

Turning to the third, ill-defined transformation, Monitor Periodic Query, classified as a Synchronous Function, the designer finds that one axiom is violated: Each function requires an incoming activator. The transformation, Monitor Periodic Query, has no means of activation. Upon reviewing the textual specification, the designer discovers that the transformation is intended to operate periodically. The specification meta-model allows this requirement to be specified directly by adding a Timer event flow, Check Furnaces, from the "System" to the transformation, Monitor Periodic Query.

In addition to correcting these specification errors, the designer also labels each transformation in the data flow diagram with a unique number. These numbers, while optional, provide a means of tracking the decomposition hierarchy, should some future version of the specification be further decomposed.

Figure 72 gives the amended data flow diagram, annotated with information inferred and elicited by CODA. The designer loads the amended specification and then, finding the classification of specification elements to be incomplete, asks CODA to classify the specification. After inquiring about the nature of terminators in the specification, CODA proceeds straight through the initial stages of classification. Upon reaching the final stage of classification, CODA makes a tentative classification of one transformation, Maintain Temperature Table, as a Synchronous Function. CODA asks the designer to confirm or override this decision. After reading the textual specification accompanying the data flow diagram, the designer decides that the transformation requires substantial execution time and should be an Asynchronous Function; thus, the designer overrides CODA's tentative classification. Also during the final classification stage, CODA finds that some application-specific knowledge can help to classify more accurately eight transformations, Determine Msg Type, Create ICP, Validate ICP, Prepare CP ACK, Wait for Query Timeout, Create New DP, Prepare IDP, and Get Temperature Reading. CODA asks the designer to supply, where known, the helpful information. In this example, the designer supplies the requested information and CODA decides how to classify each transformation in question.

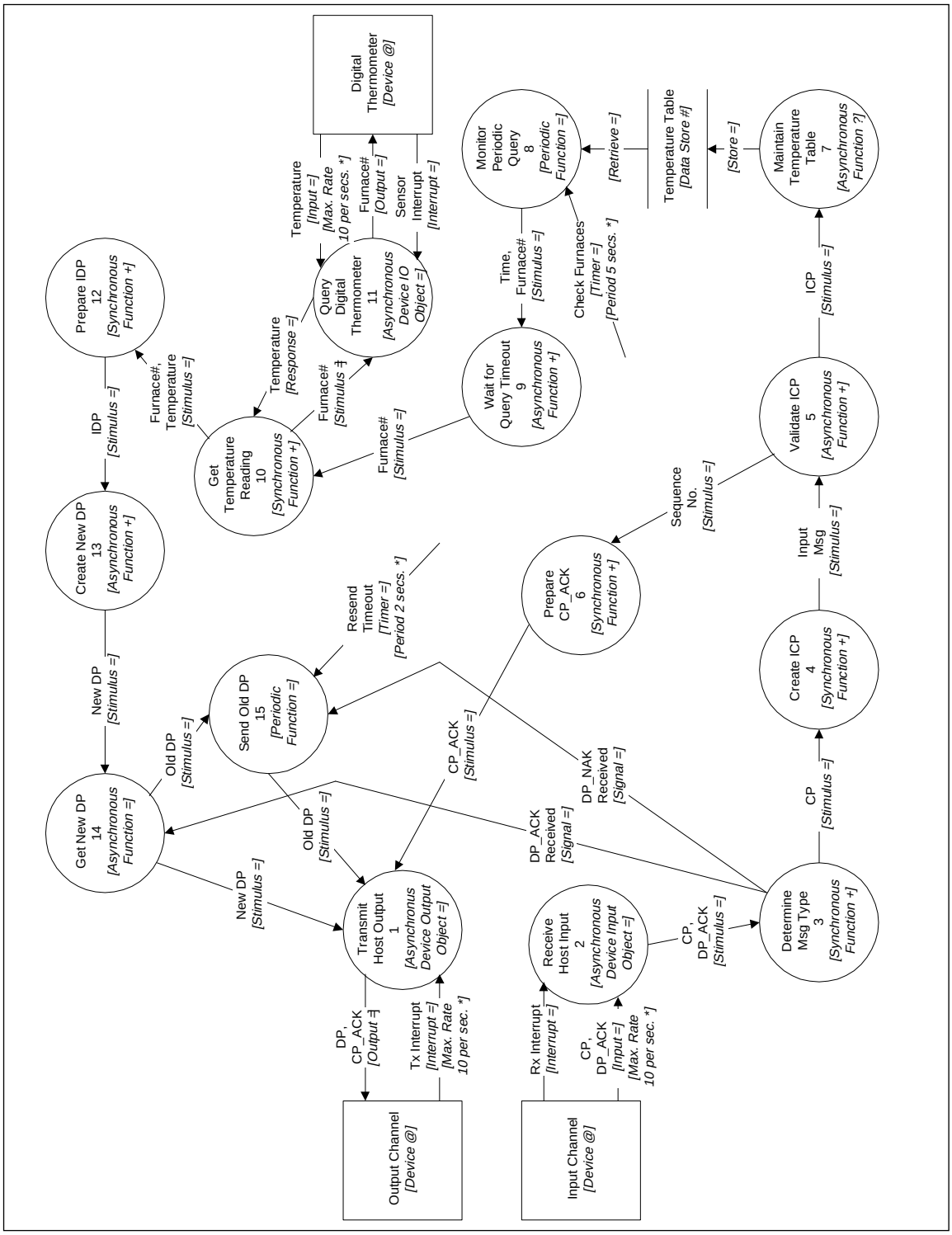

Figure 72. Amended RTS Data Flow Diagram as Analyzed by CODA

E.1.3 Eliciting Additional Information

After completing concept classification, CODA elicits additional information from the designer as required to help generate a concurrent design. Each of two timers, one for Send Old DP and one for Monitor Periodic Query, requires a period. Perusal of the textual specification indicates that the appropriate values are two and five seconds, respectively. CODA also asks the designer to supply maximum event rates for the three device-interface objects, Receive Host Input, Query Digital Thermometer, and Transmit Host Output, shown in the data flow diagram. The values supplied derive from the estimated channel rates and message sizes for each device.

Next, CODA asks the designer to supply any specification addenda that can help generate an appropriate, concurrent design. First, CODA displays a list of the asynchronous and periodic functions in the specification and asks the designer to identify any subsets of those functions that cannot execute at the same time. After examining the diagram, the designer decides that operations that update and read from the temperature table must be conducted with mutual exclusion in order to avoid reading incorrect information from the data store. For this reason, the designer indicates that two functions, Maintain Temperature Table and Monitor Periodic Query, should be placed together in an exclusion group. Second, after reading the textual specification, the designer finds that the host and remote computers use a stop-and-wait protocol to exchange data packets. This means that the remote temperature sensor either sends a new data packet, or resends an old data packet, but never performs both functions

simultaneously. For this reason, the designer indicates that the two related transformations, Get New DP and Send Old DP, should be included in an exclusion group.

To finish analyzing the specification, CODA elicits, and the designer declines to provide, any additional specification addenda. The designer verifies the specification's state as classified completely, with all axioms satisfied. From this point, a design can be generated.

E.2 Generating the Design

The designer decides to begin the design-generation process by structuring tasks from the data flow diagram. First, the designer loads a target environment description that simulates the facilities available in Ada, that is, an environment without message queuing services. The designer selects this environment because Nielsen and Shumate target their design for an Ada run-time system.

E.2.1 Structuring Tasks

Next, the designer initiates task structuring. After identifying candidate tasks, CODA attempts to allocate the remaining transformations to tasks. For a number of synchronous functions, CODA recognizes that application-specific knowledge might lead to better decisions. In these instances, CODA attempts to elicit any available insights from the designer. For example, CODA explains that a transformation, Get Temperature Reading, might be allocated to the same task as one of two, connected transformations, Wait for Query Timeout and Query Digital Thermometer. CODA then asks the designer

if the transformation in question, Get Temperature Reading, should be allocated to the same task as one or the other of the connected transformations. If the designer can provide this information, then CODA can make a better decision about allocating the transformation, Get Temperature Reading. Two other transformations, Prepare CP ACK and Determine Msg Type, each classified as a Synchronous Function, might also be allocated to one of several tasks. For each of these transformations, CODA asks the designer for, and receives, guidance.

After completing the allocation of transformations to tasks, CODA considers combining tasks and creating resource monitor tasks. Next, CODA invites the designer to review and rename tasks in the design. The designer accepts the invitation. Upon completion of task structuring, the designer saves the design and checks the state of the design process. Table 49 gives the results of CODA's task structuring, including: the tasks created, the transformations allocated to each task, and the criterion used in determining each allocation.

Table 49. Task Structuring Decisions for Remote Temperature Sensor

Task	**Transformations**	**Structuring Criterion**
Create New DP	Create New DP	Asynchronous Internal Task
Create IDP	Wait for Query Timeout Get Temperature Reading Prepare IDP	Asynchronous Internal Task User-Specified Cohesion Sequential Cohesion
Analyze Host Input	Validate IDP Prepare CP ACK	Asynchronous Internal Task User-Specified Cohesion
DT Handler	Query Digital Thermometer	Asynchronous Device I/O Task
Tx Host Msg	Transmit Host Output	Asynchronous Device I/O Task
Rx Host Msg	Receive Host Input Determine Msg Type Create ICP	Asynchronous Device I/O Task User-Specified Cohesion Sequential Cohesion
Determine Host Output	Get New DP Send Old DP	Asynchronous Internal Task Periodic Internal Task Mutually Exclusive Execution
Manage Temperature Reading	Maintain Temperature Table Monitor Periodic Query	Asynchronous Internal Task Periodic Internal Task Mutually Exclusive Execution

E.2.2 Defining Task Interfaces

After structuring tasks, the designer decides to define the interfaces between tasks in the design. CODA allocates the external interfaces for each task and for two inter-task, event flows, DP ACK and DP NAK. CODA takes these decisions without consulting the designer. Next, when allocating data flows between tasks, CODA consults with the experienced designer regarding five instances where ambiguity exists. In each instance, CODA cannot establish whether the sending task must synchronize with the receiving task. An experienced designer might be able to provide the required information. If not, then CODA makes a default decision to map each data flow to a queued message. In each of the five instances in this case study, the designer provides the missing information regarding inter-task synchronization, and CODA allocates each data flow to an appropriate message type.

CODA then detects, that at least one task receives queued messages from multiple source tasks. Given an experienced designer, CODA invites the designer to consider assigning varying priorities to appropriate queued messages. In their design, Nielsen and Shumate do not use multiple priorities; however, for this case study, the designer assigns varying priorities to queued messages received by one task, Tx Host Msg. The designer gives outgoing command-packet acknowledgments preference ahead of outgoing data packets. This choice allows a demonstration of CODA's ability to simulate priority queues when the target environment provides no message queuing services. CODA

allocates appropriate queuing mechanisms and then invites the designer to review and rename task-interface elements. The designer accepts the invitation.

E.2.3 The Task Architecture

The task architecture for the remote temperature sensor, as generated by CODA, appears as shown in Figure 73. Figure 73 depicts the state of the design after structuring tasks and defining task interfaces, but before structuring modules and integrating the task and module views.

E.2.4 Structuring Modules

To continue with the design, the designer needs to structure modules. CODA handles most module structuring decisions without consulting the designer; however, ambiguities can arise. For example, a transformation, representing a synchronous function, might be linked with one or more other transformations, previously allocated to an information hiding module. In such situations, CODA can decide to allocate a transformation to an existing module, based on sequential or functional cohesion, or can form a new module. Lacking any other information, CODA forms a new module. Given an experienced designer, however, CODA elicits any guidance the designer cares to provide. The current case study contains five, ambiguous transformations: Determine Msg Type, Create ICP, Prepare CP ACK, Get Temperature Reading, and Prepare IDP. For each of these transformations, CODA asks the experienced designer whether to include the transformation into an existing module or whether to form a new module based on the transformation. For three transformations, Determine Msg Type, Prepare

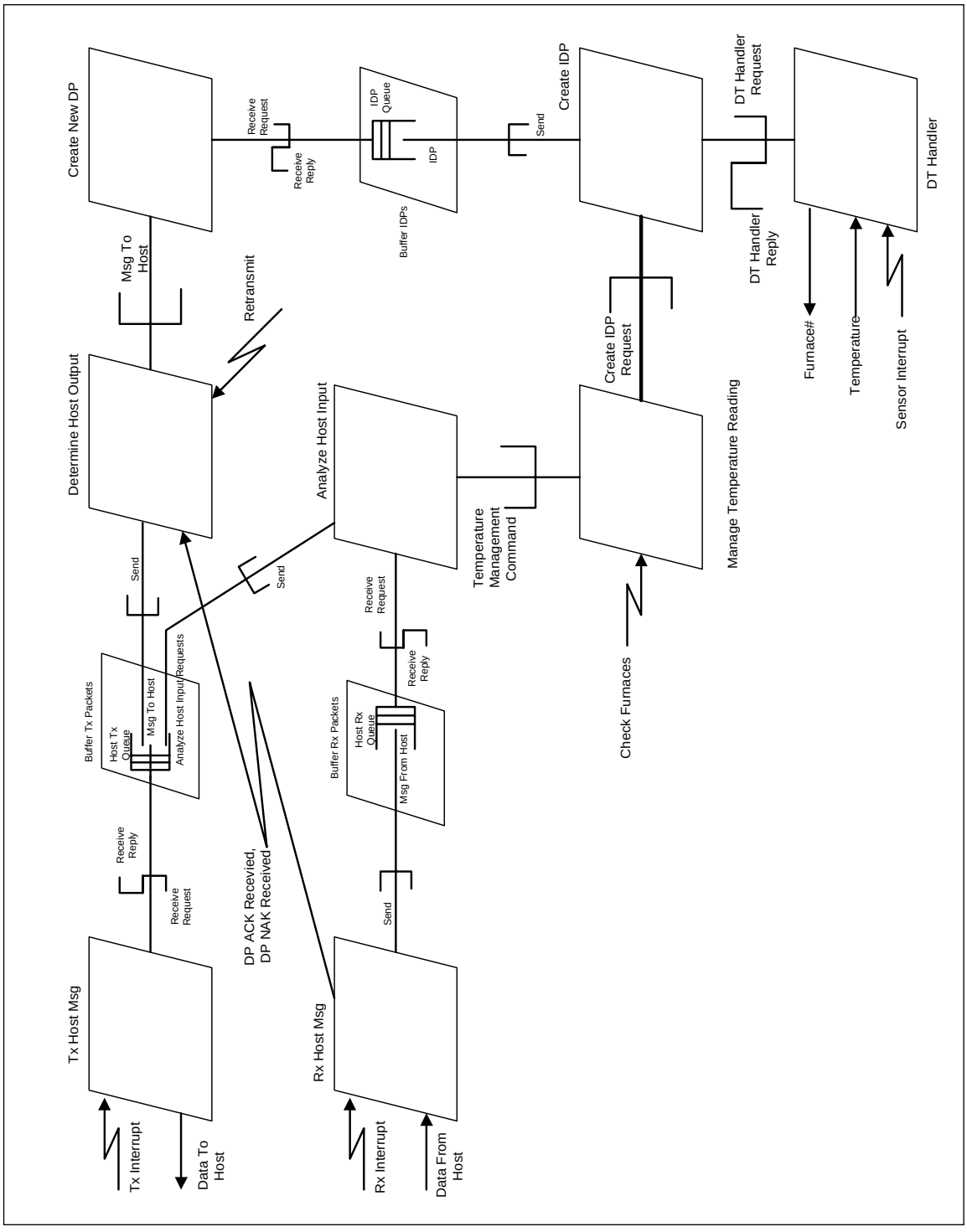

Figure 73. Task Architecture for the Remote Temperature Sensor Design

CP ACK, and Prepare IDP, the designer indicates that a new module should be formed. The designer also indicates that each of the two remaining transformations, Create ICP and Get Temperature Reading, should be included into the same module as a previously allocated transformation, Determine Msg Type and Wait for Query Timeout, respectively.

After structuring modules and determining module operations, CODA invites the designer to review and rename these new design elements. The designer accepts the invitation. Table 50 reports the results of the module structuring for the remote temperature sensor.

E.2.5 Integrating Tasks and Modules

The designer need only integrate the task and module views in order to complete the design. When asked, CODA achieves this integration without consulting the designer.

E.2.6 The Completed Design

Figure 74 depicts a software architecture diagram for the completed design, as generated by CODA. The software architecture diagram builds upon the task architecture diagram, shown previously as Figure 73, adding the modules created during module structuring. In the resulting design, tasks share no modules. Nielsen and Shumate do not generate modules; instead, because their target environment is an Ada run-time system, they employ a set of guidelines to identify Ada packages. For this reason, comparisons

between CODA's design and the design given by Nielsen and Shumate must be limited to the task architecture.

Table 50. Module Structuring Decisions for Remote Temperature Sensor

Module	**Transformation/Data Store**	**Structuring Criterion**
Temperature Table	Temperature Table Monitor Periodic Query Maintain Temperature Table	Data-Abstraction Module Read Operation Of DAM Update Operation Of DAM
Digital Thermometer	Query Digital Thermometer	Device-Interface Module
Send Message	Transmit Host Output	Device-Interface Module
Read Message	Receive Host Input	Device-Interface Module
Data Packet	Create New DP	Algorithm-Hiding Module
Temperature Checker	Wait for Query Timeout Get Temperature Reading	Algorithm-Hiding Module Designer-Allocated Function
ICP	Validate ICP	Algorithm-Hiding Module
Manage Unsent DPs	Get New DP	Algorithm-Hiding Module
Manage Sent DPs	Send Old DP	Algorithm-Hiding Module
Input Analyzer	Determine Msg Type Create ICP	Algorithm-Hiding Module Designer-Allocated Function
CP ACK	Prepare CP ACK	Algorithm-Hiding Module
IDP	Prepare IDP	Algorithm-Hiding Module

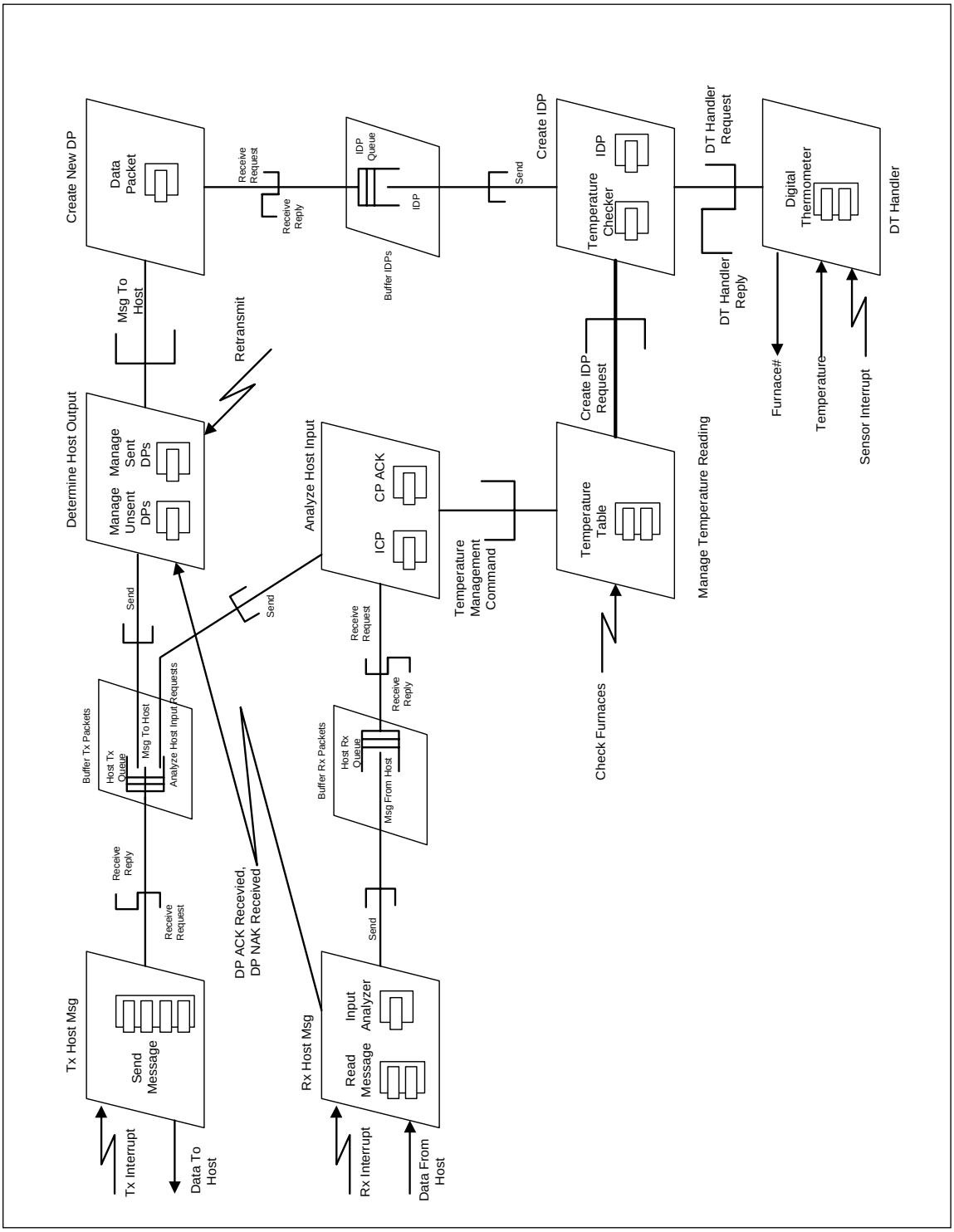

Figure 74. The Completed Design for the Remote Temperature Sensor

The task architecture given in Figure 73 aligns well with the design proposed by Nielsen and Shumate. Two essential differences appear between the two designs. First, CODA created a queued-message interface for messages received by one task, Tx Host Msg, for which Nielsen and Shumate defined two, tightly-coupled message interfaces. Because the target environment does not provide message-queuing services, CODA's decision leads to an additional, priority-queue-control task, Buffer Tx Packets, not present in the design by Nielsen and Shumate. The second difference also relates to intermediary tasks. Nielsen and Shumate identify a task to relay DP ACK signals from the Rx Host Msg task to the Determine Host Output task. CODA assumes that a target environment permitting inter-task signals provides an inherent signaling mechanism; thus, CODA does not generate a relay task for the DP ACK.[2]

E.3 CODA Unaided

When a designer cannot provide assistance in cases where CODA can benefit from such assistance, CODA takes default decisions as needed to generate a concurrent design. The data flow diagram for the remote temperature sensor, specified using Structured Analysis, leads CODA to seek the designer's assistance in many situations, during both specification analysis and design generation. If the designer could not provide any help, then what design would CODA produce from the data flow diagram for the remote temperature sensor? To answer this question, CODA generates another design

---

[2] To more closely model an Ada environment, the target environment description might indicate that no software signals are permitted between tasks. In such a case, CODA would map the events, DP ACK and DP NAK, onto a tightly-coupled message; thus, a relay task would not be generated by CODA under any circumstances.

from the amended data flow diagram, as shown in Figure 72, for the remote temperature sensor.

E.3.1 Analyzing the Specification

The analysis of the specification proceeds straightforwardly. During the classification of concepts, CODA queries the designer for assistance in classifying the terminators and the ambiguous aperiodic functions. The novice designer provides no assistance. In these cases, CODA classifies the terminators as devices and the ambiguous, aperiodic functions as asynchronous functions. Next, CODA elicits additional information. CODA forces the novice designer to provide periods for two timers, Resend Timeout and Check Furnaces, and to provide maximum rates for the three external inputs to the system. The designer must obtain this information from the textual specification accompanying the data flow diagram. CODA offers the novice designer opportunities to enter specification addenda. The novice designer provides no addenda.

E.3.2 The Completed Design

Next, CODA generates a concurrent design, using the same Ada target environment as before. In this case, CODA makes all design decisions without consulting the designer. CODA consults the designer only to elicit names for new design elements. The resulting design consists of 27 tasks, one for each of the 15 transformations on the data flow diagram and one to control each of the 12 queues required for the design, and 14 modules. One data-abstraction module consists of two transformations, Maintain Temperature Table and Monitor Periodic Query, and a data

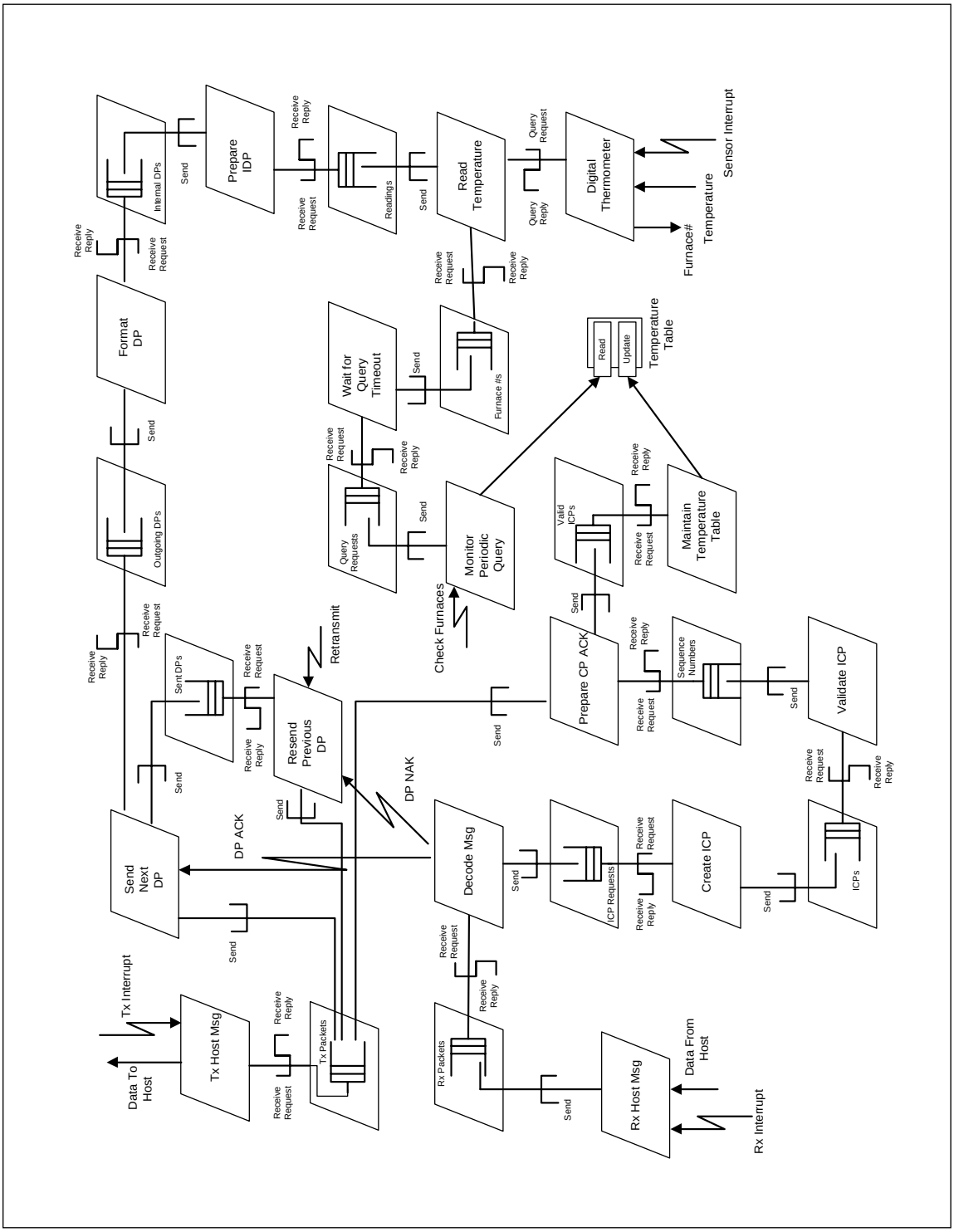

Figure 75. Design Generated, Unaided, by CODA for the Remote Temperature Sensor

store, Temperature Table. CODA allocates each of the remaining 13 transformations to an independent module. Figure 75 illustrates the task architecture for this design.

For the remote temperature system, CODA, unaided by an experienced designer during both the concept classification and the design generation, produces a less efficient design than is the case when CODA receives assistance. The design shown in Figure 75 calls for 27 tasks rather than the 11 tasks needed for the design shown in Figure 73. In addition, although not shown in Figure 75, CODA, unaided, generates a design calling for 14 modules, while, as shown in Figure 73, with help from an experienced designer, CODA created a design requiring only 12 modules. These results illustrate that CODA generates more efficient designs when a designer provides help during the classification of aperiodic functions and when an input data/control flow diagram takes advantage of the semantic concepts from the specification meta-model.